(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 504 996 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 92200730.7

(51) Int. Cl.^{5}: **G06F 7/72**

(22) Date of filing: 13.03.92

(30) Priority: 22.03.91 EP 91200639

(43) Date of publication of application:
23.09.92 Bulletin 92/39

(84) Designated Contracting States:
DE FR GB

(71) Applicant: **N.V. Philips' Gloeilampenfabrieken**
Groenewoudseweg 1
NL-5621 BA Eindhoven(NL)

(72) Inventor: **Kessels, Jozef Laurentius Wilhelmus**
c/o INT. OCTROOIBUREAU B.V. Prof. Holstlaan 6
NL-5656 AA Eindhoven(NL)

(74) Representative: **Strijland, Wilfred et al**
INTERNATIONAAL OCTROOIBUREAU B.V.
Prof. Holstlaan 6
NL-5656 AA Eindhoven(NL)

(54) Arithmetic unit for multiplying long integers modulo M and R.S.A. converter provided with such multiplication device.

(57) A systolized and modular arithmetic device has a control module, followed by a series arrangement of processing module, followed by a tail module. For multiplying an integer P and an integer Q modulo a third integer M, a provisional product is incremented each time with Q for a -1- bit in P, preceding by a doubling of the product. For a -0- bit only the doubling ensues. Normalizing mod M is effected by adding the complement of M, W, under control of propagated carry values. A similar procedure is proposed for exponentiation of Q.^F.

Rank Xerox (UK) Business Services

FIELD OF THE INVENTION

The invention relates to an arithmetic unit for multiplying long integers modulo an integer M. Generally, arithmetic units have been in use for standard word lengths, such as 8, 16 or 32 bits, and thus have been optimized for the actual word length. While operating on a word, the result is usually provided in parallel which has necessitated for complex features such as carry ripples, the complexity of such provisions and the response time incurred growing quickly with increasing word length. The present invention supports arithmetic with a clock rate that is independent of the word length.

The design is systolic in that it consists of a serial arrangement of a single control module followed by an array of processing modules, all of which are simultaneously active. Such a device is useful as a converter for the well-known RSA encryption, see R.L. Rivest et al., A Method for obtaining Digital Signatures and Public-Key Cryptosystems, Comm. ACM, Vol. 21 (February 1978), pages 120-126. The RSA encryption is based on the recognition of a triad of integers E, D, M, such that for any integer message X < M: (X**(E*D))mod M = X. Herein, the asterisk conventionally indicates a multiplication, the double asterisk an exponentiation. Both E and D are smaller than M. In a typical field of use, M consists of 512 bits. The algorithm is used in public cryptographic systems, especially, because the encoding key (E, M) and the decoding key (D, M) cannot be practically derived from each other. Raising to a particular power can be done by repeated multiplication with the starting integer X.

The multiplication of non-integer messages is being considered an obvious extension. If N is the number of bits of M, conventionally the complexity of the above power raising is of the order of O(N3), which leads to a low conversion speed in case of execution on a sequential general purpose machine even if the latter is intrinsically fast.

SUMMARY OF THE INVENTION

Among other things, it is an object of the invention to provide a systolic machine for multiplying two natural numbers P, Q modulo a third natural number M with a machine size of O(N) sufficient to accommodate P, Q, M, and a high throughput speed. Conventional solutions have two problems. First a multiplication phase followed by a separate reduction phase would lead to an intermediate result of size 2*N in between the two phases. By now unconventionally handling the bits of the control operand in the multiplication procedure in order of decreasing significance, the operations in the multiplication procedure and the operations in the reduction procedure can be interleaved so as to keep the intermediate result restricted. The interleaving of both procedures leads to a dynamic control sequence. Therefore an arithmetic unit is needed with control signals to indicate which operation is to be executed. The second problem is that in such an arithmetic unit with large size, both the propagation of carries and the broadcasting of the control signals would prevent a high clock rate. Therefore in the systolic solution carries ripple in the direction of increasing significance (carry save technique) while control signals move in the opposite direction.

It is the above way of cross-wise propagation of the necessary ingredients to the calculation (apart from those quantity parts that are locally present) that allows the set-up to be described hereinafter to attain a throughput speed of O(1/N). According to one specific aspect of the invention the object is realized by providing a systolized and modular arithmetic device for multiplying a first multibit integer Q with a second multibit integer P modulo a third multibit integer M, said arithmetic unit comprising a control module followed by a series arrangement of processing modules, followed by a tail module, said processing modules having modular storage means for storing mutually exclusive first bit parts of said first integer Q and mutually exclusive second bit parts indicating said third integer M of pairwise equal significance and along said series arrangement of monotonously decreasing significance levels away from said control module, said control module having presentation means for as based on successive bit positions of said second integer P presenting a control bit string for by each control bit so present in a first cycle part of each cycle rippling an elementary multiplication operation in the low significant direction, through said series arrangement, in a second cycle part rippling a carry propagation in the high significant direction, in a third cycle part selectively upon detecting an egression over said third integer rippling a modularizing operation in the low significant direction and in a fourth cycle part rippling a borrow quantity in the high significant direction, four successive cycle parts constituting a complete cycle associated with a particular bit position of said second multibit integer P, and wherein said tail module has emulating means for emulating dummy parts of said first and third integers with respect to the low significant end of said series arrangement.

Although the general set-up works excellent according to this general principle, it has been considered better to translate the modularizing into an addition with the complement of M. In this respect, advantageously, the control module has presentation means for serially presenting a control signal string for forward propagation through said series arrangement, under control of successive bit positions of said second integer according to monotonously decreasing significance levels, to wit a -double- instruction followed by an addQ instruction under control of a -1- bit in P, but only a -double- instruction under control of a zero bit in P, and said series arrangement having backpropagation means for backpropagating a carry signal after each control signal, the control module forward propagates an addW instruction for every carry received which results effectively in a substraction of M. Advantageously, the normalization value W is the complement of said third integer M (W= 2**N-M, where N is the size of M, i.e. 2**(N-1) -1 < M < 2**N).

Advantageously, each processing module operates on only a single bit significance level. This provides an extremely simple layout of the modules, whereas the granularity allows for using only the number of modules needed. Alternatively, modules could operate on a succession of bit significance levels, such as 2, 4 or 8. Within each module, the operation may be conventional, so that the module operates faster than would be able with the corresponding single-bit-modules. Between successive modules, the inventive procedures and hardware provisions would be presented. This would still allow for enough flexibility in most cases, while still keeping the design of the modules relatively simple.

The invention also relates to a device for exponentiation of an integer X to a power E modulo said third integer M, wherein said series arrangement has control means for loading said integer X as representing said first integer Q and said control module has second control means for activating said presentation means for presenting various powers of said integer X as representing said second integer P, and wherein furthermore recycling means are provided to recycle a preliminary product back to said control module as representing a subsequent value of said second integer P. Exponentiation is a sequence of multiplications, wherein the two factors are built from the original integer X. In an elementary set-up, only the product is recycled and multiplied by X, so that the exponent is raised by one.

For faster operation, however, either the preliminary product is recycled and multiplied with itself, followed by a multiplication by X, or only the multiplication with itself is executed, as based on whether the exponent bit in question is 1 or 0, respectively. The only necessary provision is that the storage capacity per module is sufficient, inasmuch now always the one factor Q, the product actually being formed, and the preliminary product of the preceding multiplication must be stored. For example, for E=11 (binary 1011), the first bit produces X, the second $X.X=X^2$, the third $X^2.X^2.X=X^5$, the fourth $X^5.X^5.X=X^{11}$. Note that the first bit in fact does $X^0.X^0.X=X$

Various advantageous aspects are recited in dependent Claims.

BRIEF DESCRIPTION OF THE FIGURES

A preferred embodiment of the invention will be explained in particular with respect to the accompanying Figures and Tables, wherein first the general setup is explained with respect to an embodiment, next an embodiment of the control module, thereafter an embodiment of the processing module.

Now, Figures 1, 2, 3a, 3b are various symbolic diagrams of multiplication/exponentiation devices while also illustrating a 3-bit multiplication cum normalization modulo M.

Tables 1-12 specify various operations and definitions.

DESCRIPTION OF A PREFERRED EMBODIMENT

Figure 1 is an elementary block diagram of an exponentiation/multiplication device according to the invention. Herein, the device is conceptually decomposed into a control module R and an array of processing modules symbolized by a block SA_N, wherein N is the number of modules. The control module receives from the outer world, such as a higher level host machine, signals H_{in}, V_{in} and outputs thereto through V_{out}. Towards the array of processing modules the control module outputs signal F_{out} and receives therefrom signals C_{in}, P_{in}. Likewise, the array of processing modules receives signals F_{in} and outputs signals C_{out}, P_{out} back to the control module. Between the processing modules the interface is the same as the one between control module and the string of processing modules.

The setup shown is detailed hereinafter for the exponentialization. Multiplication is a subcomputation of exponentiation. Now, the data available in the control module consists of E, the exponent value that is smaller than M, and two counters n0, n1 in the range [0,N] (where N is the number of bits of M). The array of processing modules must now calculate (X ** E) mod M. To this effect the array of processing modules stores the following quantities:

W=2**N-M and X the quantity that is to be raised to exponent E. Moreover the processing modules have means to store the intermediate results P,Q and r. More particularly, Q is an intermediate result of exponentiation, whereas r is an intermediate result of the multiplication of the operands P and Q (where P is either X or Q).

DESCRIPTION OF THE OPERATIONAL PROCESS

Table 1 specifies the types of various quantities figuring in the method. The phase is a binary value. At the first value the key is loaded. At the second value the message blocks are loaded and converted and finally output. The type bit is standard binary. The type carry is a three-valued carry quantity. The type instruction may have any of eleven values, each named as according to the list thereafter. In particular, to be discussed hereinafter in detail, -store- means transferring from the storage space reserved for quantity r to the storage space reserved for the quantities W,Q,X, and P (storeW, storeQ, storeX, storeP). The instruction may be coded in a conventional way such as a four bit quantity. Non-used values may be designed to control special functions, such as test.
The instruction loadX transfers information from X to r. Next, the channel types are specified. H_{in} is the externally controlled phase, V_{in}, V_{out}, P_{in} are binary channels, C_{in} is a channel of type carry, and F_{out} for outputting control values of type instruction. The control module receives carries through input port C_{in}. In the communication between the control module and the array of processing modules the communications through the instruction channel F_{out} and the carry channel C_{in} alternate. This section, only considers the operation of the control module on a functional level. On a hardware level, mapping of those functions on elements of a programmed microprocessor would be conventional. Alternatively, a translation to dedicated special purpose hardware would be obvious to the skilled art technician. Generally, several modules could be realized on an integrated circuit. Alternatively, the whole device could be computer-assisted-designed into a single integrated circuit.

Now, for the control module in particular, Table 2 gives the variable declaration and the principal execution loop by the control module. There is declared a variable h of type -phase-, two carry values car and c of type carry, two count values of type (0..N), a bit value b and an array E of N bits containing the exponent.

The infinite loop waits for the phase control signal which it receives through channel H_{in} in variable h. If the phase value "loadkey" (actually a binary value) is received, the conversion key is loaded. The key, in particular consists of quantities E, W, to wit, the exponent and the complemented modulo value M. First the exponent is loaded in the bit array E, then the value W is loaded in the distributed variable r(r is distributed over the processing modules). Subsequently this value is stored in distributed variable W by forwarding the instruction storeW to the array of processing modules. The storeW instruction is followed by a carry communication from the processing modules to the control module.

On the other hand, if the received phase indicates "convert" the message block X is loaded in variable r and subsequently stored in X by forwarding the instruction storeX and waiting for the carry value. Subsequently, the procedure X_exp_E_mod_M is called, which yields a value that is smaller than 2N. Since the result must be smaller than M, the normalize operation is executed which yields an equivalent value (modulo M) smaller than M. Finally the result is output. The exponentiation and normalization effectively are done by the series of processing modules to be detailed hereinafter. Inasmuch as h may have only two different values always one of the phases prevails.

In the control module, via V_{in}, V_{out} and P_{in} always bit-serially blocks of N bits are communicated while starting with the most significant bit. Table 3 gives the realizations of the two operations load_exponent and load_r. In load_exponent, count n0 is initialized to N. Thereafter, the system loops in that for each bit position, the bit on signal V_{in} is awaited and stored in an array E. In operation loadr, the variable r is first initialized to zero by issuing the instruction set0 and thereupon, the carry c is awaited on C_{in}. Next, the count n0 is set to N. Thereafter the control module executes a loop in which for each bit position, first, the bit on port V_{in} is received and the count decremented. Next, the value of r is doubled by issuing the instruction mul2 and awaiting carry c from C_{in}. If the bit received through V_{in} is equal to zero the iteration step is ready. If the received bit is equal to 1, the value of r is incremented by one by issuing the instruction add1 and once more receiving a carry from C_{in}. After N such steps r has the value received through port V_{in}.

The procedure used to compute (X**E) mod M is based on the algorithm shown in Table 6. The exponentiation maintains the invariant
(Q**2**n) * X**(E mod 2**n) = X**E. First Q is set to 1 and n is set to N. Now, the most significant zero bits in the exponent are skipped. Thereafter, as long as n is positive, the following loop is executed. First, n is decremented and Q is squared. Next, if the current bit in the exponent equals 1, Q is multiplied by X; otherwise Q is not modified.

Likewise, Table 7 gives the algorithm for multiplication, i.e. for computing (P*Q). The computation maintains invariant (P mod 2**n) * Q + r * (2**n) = P * Q. In this case variable r is set to zero; and variable n is set to N. Now, as long as n is positive, first, n is decremented and r is doubled. Next, depending on which of the two guards is true, either no operation is effected, or Q is added to r.

Table 4 shows the procedure

X_exp_E_mod_M as executed by the control module. The distributed variable Q is set to 1 by issuing three instructions. First, the instruction set0 is output and carry c awaited, this makes r equal to zero. Next, instruction add1 is outputted and carry c awaited. Thereafter, instruction storeQ is outputted and the carry awaited. The combination of these three operations sets variable Q to 1. Next counter n0 is initialized to N. Thereafter the most significant zero bits in the exponent are skipped. The remaining bits of the exponent are handled by executing a loop in which each step starts in a state in which the variables r and Q contain the same value. In each step first the procedure mul_Q_mod_M is executed, which operation is specified hereinafter. This procedure, in particular, assigns to both r and Q the value r * Q mod M. Therefore, since r was equal to Q at the beginning of the procedure execution, the procedure has squared Q modulo M. Next, for zero value of the exponent bit pointed to by n0, the iteration step is completed. For a one value of the exponent bit in question, first, the variable r is set to X by issuing the instruction loadX and subsequently receiving a carry. Subsequently the procedure mul_Q_mod_M is executed, which results in the computation of Q*X modulo M.

Table 5 specifies the procedure mul_Q_mod_M. In the power raising, P is an auxiliary variable for the multiplication. First, the value in r is stored in variable P by issuing the instruction storeP and subsequently receiving a carry. Next, the instruction set0 is output and the carry c is awaited. Now, the quantities car and b are made zero, and count n1 is set equal to N. It should be noted that n0 sequences along the bits of the exponent value, whereas n1 sequences along the bits of multiplication factor P. Next, a loop is executed as long as either the car value, or b or n1 is positive (car>0 or b=1 or n1>0). In each loop step one out of three commands is executed. If the car value is positive (car>0), the value of car is decreased by one and an addW instruction is issued. Both operations together effectuate a subtraction by M, since a carry represents the value 2**N and M=2**N-W. If the car value is zero and b equals one (car=0 and b=1), b is set to zero and the addQ instruction is output. If both car and b are zero (car=0 and b=0), n1 is positive (follows from loop condition) and therefore the next bit of factor P can be received and count n1 decremented by one. Next, the mul2 instruction is output. After issuing one of the three instructions, the carry c is awaited and added to the value of car. In this way all bits of factor P are treated in succession. At the end of the loop r is equivalent to P*Q modulo M, but it is not necessarily smaller than 2**N. Therefore, a new loop is entered in which all possible carries in the processing modules are propagated towards the control module. First count n1 is again set to N. The loop continues as long as the value of either car or n1 is positive. If car is positive, car is decremented, count n1 is set to N, and instruction addW is output. If car is equal to zero (which implies that n1 is positive), count n1 is decremented and the ident instruction is output. The instruction ident does not modify the value of r, it only serves to propagate carries. After issuing one of the two commands, carry c is awaited and added to the value car.

After the loop the value of r is smaller than 2**N. The value of r is then stored in Q by issuing the storeQ instruction and receiving a carry.

After the exponentiation the value of r is normalized. Table 8 details the normalizing operation, which assigns to variable r the value r mod M. Initially, r is smaller than 2**N and after normalization r is even smaller than M. Initially the value of r is also present in Q. First the value of W is added to r by issuing the addW instruction and subsequently propagating all carries. If a carry is received, the original r was larger than M and the updated value is r-M. If on the other hand, no carry is received, the old value of r, still being present in Q, was correct and should be output. So, the operation starts with sending the instruction addW and awaiting a carry. Count n1 is set to N. Now, as long as c=0 and n1 is non-zero, the ident instruction is output, the counter n1 is decreased, and a carry value received. Upon leaving the loop one of the following conditions holds. If no carry is received (c=0), the value of Q is restored in r by first issuing the instruction set0 and receiving a carry value and subsequently issuing the instruction addQ and receiving a carry value. If on the other hand, the carry is equal to one, the updated value of r is correct and is left unmodified. Finally, the result is output as indicated in Table 9. First, the value of r is stored in the shift register by issuing the storeP instruction and awaiting a carry value. Next, count n0 is set equal to N. In the next following loop, as long as n0 > 0, a bit is received from P_{in} and subsequently output through V_{out}, after which n0 is decremented.

Figure 2 more specifically shows the systolic design of SAn. The design is recursive in that module SAn+1 consists of an arithmetic cell A, a shift register cell S, and a similar but smaller module SAn. If n is positive, module SAn again consists of a pair of A/S cells and a module SAn-1. Finally there is a tail cell SA0, which does not effectively execute any arithmetic.

Table 10 exemplifies the function of tail cell SA0. It continuously loops while receiving instruction f, and depending on the nature of this instruction, outputs either a 1 or a 0 on its carry output

C_{out}. SA0 never communicates through port P_{out}. Table 11, likewise, shows the functions of shift register cell S. It continuously loops and in each loop step it first waits for a bit either through port in from the corresponding arithmetic cell (load operation) or through port P_{in} from its right-hand neighbour (shift operation). The bit received is subsequently output to its left neighbour through port P_{out}. In fact, the cell is a one bit shift register with a multiplexed input. Note that at any time at most one of the two input ports offers a bit.

Likewise, Table 12 shows the functions of arithmetic cell A .It has a variable c of type carry, a variable carry with a value range 0..4; a variable f of the type instruction, and three bits w, q, x. Now, first the cell awaits an instruction from input F_{in}; at reception, variables car, c are reset to zero. Next, a loop is executed where the the instruction value determines which operation is executed. For the eleven instructions listed earlier the following operations are performed: set0 sets car to zero, add1 retains the value of car, mul2 multiplies car by two, ident retains the value of car, addW adds w to car, addQ adds q to car, loadX assigns to car the value of x, storeW assigns to w the least significant bit of car (car mod 2), storeQ assigns to q the value (car mod 2), storeX assigns to x the value (car mod 2), storeP outputs the value (car mod 2) through output port out. Next, the instruction f is output through output port F_{out}; and the two most significant bits of car are output through Cout. Then, the next instruction is received via Fin and a carry value via C_{in}. The carry value received is added to the least significant bit of car and then the next loop step is executed.

The systolic design is extreme in that each processing module contains one bit of the values P, Q and W. Such a solution is fast, but make great demands on area and power consumption, since such a bit-slice architecture requires the maximum number of cells to construct the multiplier. If such a high speed is not required, a slower but cheaper design can be obtained by increasing the slice size.

For completeness' sake, Figures 3a, 3b give a complete example of multiplying a threebit integers P=4(100) by Q=5(101) modulo M, represented by W=2 (010). Control module RS, processing modules H2, H1, H0 have been shown, together with the quantities they contain. In Figure 3b, the first column shows 37 states, which each correspond either to the operation in a cycle part (indicated by arrows), together with the quantities transferred, or by the situation after such transfer and the operation ensued. In consequence, a sequence of 8 lines corresponds to one cycle. The quantity n is the number of bits awaiting processing, true and false have been abbreviated as t, f, respectively, d is the binary produced by the tail module emulating a dummy quantity, id (ident), aw (addw), aq (addq), db (double) are self explanatory. On line 37 the control module finds that d=true, indicating the end. Given the bit length of the two multiplicands, this could come no later than line 37, so it may as well be determined by counting. Dependent on the various quantities W, P, Q, the end could also come earlier, so the embodiment shown is somewhat faster, under particular circumstances.

Claims

1. A systolized and modular arithmetic device for multiplying a first multibit integer Q with a second multibit integer P modulo a third multibit integer M, said arithmetic unit comprising a control module followed by a series arrangement of processing modules, followed by a tail module, said processing modules having modular storage means for storing mutually exclusive first bit parts of said first integer Q and mutually exclusive second bit parts indicating said third integer M of pairwise equal significance and along said series arrangement of monotonously decreasing significance levels away from said control module, said control module having presentation means for presenting a control bit of a control bit string in a first cycle and receiving means for receiving a carry value in a second cycle part, and wherein said processing modules have means to receive simultaneously a control bit from said control bit string from their respective more significant neighbour and a carry value from their respective less significant neighbour in a first cycle phase, as well as presentation means for presenting both a carry value to their respective more significant neighbour and the control bit of said control bit string to their respective less significant neighbour in a second phase, and wherein each processing module operates half a cycle out of phase with respect to its neighbours, and wherein said tail module has emulating means for emulating dummy parts of said first and third integers with respect to the low significant end of said series arrangement.

2. A device as claimed in Claim 2, wherein said control module has presentation means for serially presenting a control signal string for forward propagation through said series arrangement, under control of carries received as well as successive bit values of said second integer according to monotonously decreasing significance levels, to wit a double instruction followed by an addQ instruction under control of

a -1- bit in P, but only a double instruction under control of a zero bit in P, and said series arrangement having backpropagation means for back propagating a carry signal after each control signal, the control module forward propagates an addW instruction for every carry received which results effectively in a substraction of M.

3. A device as claimed in Claim 1 or 2, wherein each processing module operates on a single bit significance level.

4. A device as claimed in Claim 1, 2 or 3, wherein said control module has detection means for after presentation of all bit signals of said second integer detecting arrival of an unchanged dummy signal from said tail module signalling that no carry or non-dummy control signal is being propagated in said series arrangement for so signalling availability of a modularized product.

5. A device as claimed in Claim 1, 2 or 3, wherein said control module has count means for after presentation of all bit signals of said second integer counting said cycles for upon reaching a particular count as determined by the length of said series arrangement signalling that any carry or non-dummy control signal had been rippled out for thereby signalling availability of a modularized product.

6. A device as claimed in Claim 1 for exponentiation of an integer X to a power E modulo said third integer M, wherein said series arrangement has control means for loading said integer X as representing said first integer Q and said control module has second control means for activating said presentation means for presenting various powers of said integer X as representing said second integer P, and wherein furthermore recycling means are provided to recycle a preliminary product back to said control module as representing a subsequent value of said second integer P.

7. A device as claimed in Claim 6, said control module having selection means for under control of successive bits in said exponent, either squaring said preliminary product, or squaring said preliminary product, followed by multiplying the new preliminary product by said integer X.

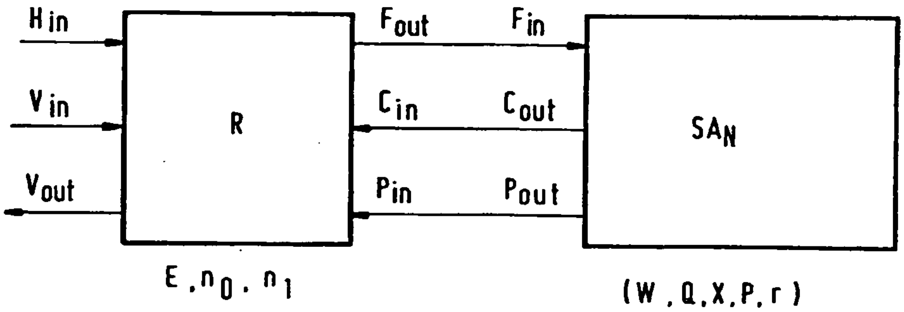

FIG.1

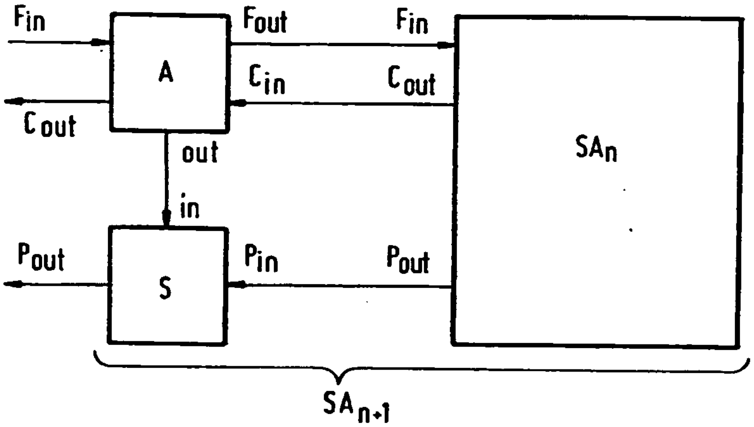

FIG.2

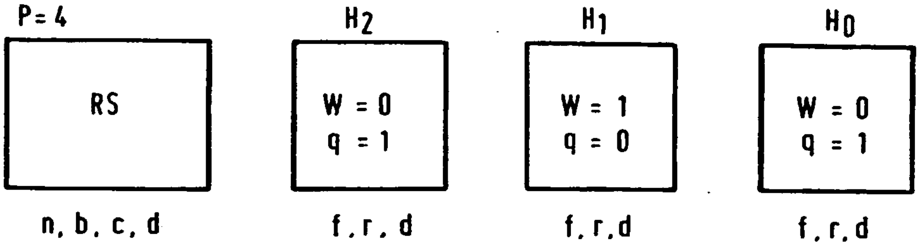

FIG.3a

```
1.  3, 0, 0 true    id, 0, true    id, 0 true    id, 0, true
2.              db→
3.  2, 1, 0, false db, 0, false
4.              ←0,f          db→
5.  2, 1, 0, f     db, 0, f      db, 0, f
6.              aq→           ←0,f          db→
7.  2, 0, 0, f     aq, 1, f      db, 0, f      db, 0, f
8.              ←0,f          aq→           ←0,f          db→
9.  2, 0, 0, f     aq, 1, f      aq, 0, f      db, 0, f
10.             db→           ←0,f          aq→           ←0,t
11. 1, 0, 0, f     db, 2, f      aq, 0, f      aq, 1, f
12.             ←1,f          db→           ←0,f          aq→
13. 1, 0, 1, f     db, 0, f      db, 0, f      aq, 1, f
14.             aw→           ←0,f          db→           ←0,t
15. 1, 0, 0, f     aw, 0, f      db, 0, f      db, 2, f
16.             ←0,f          aw→           ←1,f          db→
17. 1, 0, 0, f     aw, 0, f      aw, 2, f      db, 0, f
18.             db→           ←1,f          aw→           ←0,t


19. 0, 0, 0, f     db, 2, f      aw, 0, f      aw, 0, f
20.             ←1,f          db→           ←0,f          aw→
21. 0, 0, 1, f     db, 0, f      db, 0, f      aw, 0, f
22.             aw→           ←0,f          db→           ←0,t
23. 0, 0, 0, f     aw, 0, f      db, 0, f      db, 0, f
24.             ←0,f          aw→           ←0,f          db→
25. 0, 0, 0, f     aw, 0, f      aw, 1, f      db, 0, f
26.             id→           ←0,f          aw→           ←0,t
27. 0, 0, 0, f     id, 0, f      aw, 1, f      aw, 0, f
28.             ←0,f          id→           ←0,f          aw→
29. 0, 0, 0, f     id, 0, f      id, 1, f      aw, 0, f
30.             id→           ←0,f          id→           ←0,t
31. 0, 0, 0, f     id, 0, f      id, 1, f      id, 0, t
32              ←0,f          id→           ←0,t          id→
33. 0, 0, 0, f     id, 0, f      id, 1, t      id, 0, t
34.             id→           ←0,t          0,t→          ←0,t
35. 0, 0, 0, f     id, 0, t      id, 1, t      id, 0, t
36.             ←0,t          id→           ←0,t          id→
37. 0, 0, 0, t     id, 0, t      id, 1, t      id, 0, t
```

FIG. 3b

```
type phase {load key, convert}
     bit   (0..1)
     carry (0..2)
     instruction {set0, add1, mul2, ident, addW,
                  addQ, loadX, storeW, storeQ, storeX,
                  storeP}
     $H_{in}$ : phase
     $V_{in}$, $V_{out}$, $C_{in}$, $P_{in}$ : bit
     $F_{out}$: instruction
```

Table 1

```
Control module R
var h:= phase;
    car, c: carry;
    $n_0$, $n_1$ (0..N)
    b: bit
    E: array [0..N-1] of bit

do  true   $H_{in}$? h;
    if h = loadkey    load_exponent;
                      load_r;
                      $F_{out}$! storeW; $C_{in}$? c
    [] h = convert    load_r;
                      $F_{out}$! storeX; $C_{in}$? c
                      X_exp_E_mod_M;
                      Normalize;
                      output
    fi
od
```

Table 2

```
load_exponent: $n_o$=N;
            do $n_o$ > 0    $V_{in}$? b;
                       $n_o$:=$n_o$-1; E[$n_o$]:=b
            od

load_r : F! set0; $C_{in}$? c
        $n_o$:=N;
        do $n_o$ > 0    $V_{in}$? b; $n_o$:=$n_o$-1;
                   $F_{out}$! mul2; $C_{in}$? c;
                   if b=0      skip
                   [] b=1      $F_{out}$! add1; $C_{in}$? c
                   fi
        od
```

Table 3

```
X_exp_E_mod_M:
$F_{out}$! set0; $C_{in}$? c;
$F_{out}$! add1; $C_{in}$? c;
$F_{out}$! storeQ; $C_{in}$? c;
$n_o$:=N;
do E [$n_o$-1]=0    $n_o$:=$n_o$-1 od;
do $n_o$ > 0    $n_o$:=$n_o$-1;
           $F_{out}$! set0; $C_{in}$? c
                                r:=Q
           $F_{out}$! addQ; $C_{in}$? c
           mul_Q_mod_M;
           if E [$n_o$]=0    skip
              E [$n_o$]=1    $F_{out}$! loadX; $C_{in}$? c;
                         mul_Q_mod_M
           fi
od
```

Table 4

```
mul_Q_mod_M:
$F_{out}$! storeP; Ci? c;
$F_{out}$! set0; Ci? c;
car:=0; b:=0; $n_1$:=N;
do car > 0 or b=1 or $n_1$ >0
        if car > 0    car:=car-1; $F_{out}$! addW
        [] car=0 and b=1    b:=0; $F_{out}$! addQ
        [] car=0 and b=0    $P_{in}$? b; $n_1$:=$n_1$-1;
                            $F_{out}$! mul2
        fi;
        $C_{in}$? c; car:= car+c
od;
$n_1$:=N
do car > 0 or $n_1$ > 0
   if car> 0     car:=car-1; $n_1$:=N; $F_{out}$! addW
   [] car=0      $n_1$:=$n_1$-1; $F_{out}$! ident
   fi;
   $C_{in}$? c; car:=car+c
od;
$F_{out}$! storeQ; $C_{in}$? c
```

Table 5

Exponentiation:

```
Q:=1; n:=N;
do E [n-1]=0     n:=n-1 od;
do n > 0          n:=n-1;
              Q:= Q*Q;
              if E [n]=0     skip
                 E [n]=1     Q;=Q*X
              fi
od
```

Table 6

Multiplication:

```
r:=0; n:=N;
do n > 0   n:=n-1;
           r:=2*r
           if P[n]=0     skip
              P[n]=1     r:=r+Q
           fi
od
```

Table 7

Normalize:

```
$F_{out}$! addW; $C_{in}$? c;
$n_1$:=N;
do c=0 and $n_1$ > 0     $n_1$:=$n_1$-1
                         $F_{out}$! ident; $C_{in}$? c
od;
if c=0    $F_{out}$! set0; $C_{in}$?c
          $F_{out}$! addQ; $C_{in}$?c
[] c=1    skip
fi
```

Table 8

Output:

```
  $F_{out}$! storeP; $C_{in}$? c;
  $n_o$:=N;
  do $n_o$ > 0    $P_{in}$? b; $V_{out}$! b; $n_o$:=$n_o$-1 od
```

Table 9

```
Tail cell   $SA_0$:
  do true   $F_{in}$? f;
            if f=add1     $C_{out}$! 1
            [] f = add1   $C_{out}$! 0
            fi
  od
```

Table 10

```
Shift register cell S:
do true   if in? b
             $P_{in}$? b
          fi;
          $P_{out}$! b
od
```

Table 11

```
Arithmetic cell A
var c: carry; car:(0..4); f: instruction
    w,q,x : bit
$F_{in}$? f; car:=0; c:=0;
do true   if f=set0     car:=0
        [] f=add1     car:=car
        [] f=mul2     car:=2*car
        [] f=ident   car:=car
        [] f=addW     car:=car+w
        [] f=addQ     car:=car+q
        [] f=loadX   car:=x
        [] f=storeW   w:=(car mod 2)
        [] f=storeQ   q:=(car mod 2)
        [] f=storeX   x:=(car mod 2)
        [] f=storeP  out!(car mod 2)
        fi;
        $F_{out}$! f; $C_{out}$! (car div 2)
        $F_{in}$? f; $C_{in}$? c; car:=(car mod 2)+c
od
```

Table 12